arandomness Documentation

Release 0.1.0rc6

Alex Hyer

Nov 17, 2017

Contents

1 Introduction **3**

2 Installation **5**

3 Contents **7**
- 3.1 argparse . . . 7
- 3.2 string . . . 12
- 3.3 trees . . . 13

4 Indices and tables **17**

5 Copyright **19**

Python Module Index **21**

Authors Alex Hyer

Date Nov 17, 2017

Version 0.1

Initializes arandomness package

CHAPTER 1

Introduction

arandomness is a package containing modules that I find myself re-writing for many programs. I've organized these modules into this amalgamative package as they are generally too diverse to fit into independent libraries. In general, I find the modules in arandomness to be very useful for many unrelated applications and wished to have them readily accessible in a unified, production-level library with proper unit tests. I hope you find something in this library useful!

CHAPTER 2

Installation

```
pip install arandomness
```

CHAPTER 3

Contents

3.1 argparse

Initializes argparse package of arandomness

Copyright: __init.py__ initializes argparse package of arandomness Copyright (C) 2017 Alex Hyer

> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

3.1.1 Introduction

The `argparse` subpackage of arandomness contains scripts and actions to expand the utility of Python's argparse library.

3.1.2 CheckThreads

`CheckThreads` is an `argparse` `action`, as such, it is called as the value of the `action` argument in `argparse`. For example:

```
from arandomness.argparse import CheckThreads
import argparse
parser = argparse.ArgumentParser(description=__doc__)
parser.add_argument('-t', '--threads',
                    action=CheckThreads,
```

```
                    type=int,
                    default=1,
                    help='number of threads to use')
args = parser.parse_args()
```

When `-t` is parsed, the value is passed to `CheckThreads` which then checks that the value is between 1 and the maximum number of threads on the computer as per multiprocessing.cpu_count().

API Documentation

class `arandomness.argparse.`**`CheckThreads`**(*option_strings*, *dest*, *nargs=None*, ***kwargs*)

Argparse Action that ensures number of threads requested is valid

`__call__`(*parser*, *namespace*, *values*, *option_string=None*)

Called by Argparse when user specifies multiple threads

Simply asserts that the number of threads requested is greater than 0 but not greater than the maximum number of threads the computer can support.

Parameters

- **`parser`** (*`ArgumentParser`*) – parser used to generate values
- **`namespace`** (*`Namespace`*) – parse_args() generated namespace
- **`values`** (*`int`*) – actual value specified by user
- **`option_string`** (*`str`*) – argument flag used to call this function

Raises

- `TypeError` – if threads is not an integer
- `ValueError` – if threads is less than one or greater than number of threads available on computer

3.1.3 CopyRight

`CopyRight` is an `argparse action` that simply takes in text, strips it of leading and trailing whitespace, prints it, and exits the program. Its functionality is analogous to `argparse`'s version. The action can take in arbitrary text and is only named `CopyRight` for code readability.

```
from arandomness.argparse import CopyRight
import argparse
parser = argparse.ArgumentParser(description=__doc__)
parser.add_argument('--copyright',
                    action=CopyRight,
                    copyright_text='This is my copyright',
                    help='print copyright and exit')
args = parser.parse_args()
```

API Documentation

class `arandomness.argparse.`**`CopyRight`**(*option_strings*, *dest*, *copyright_text=None*, *nargs=None*, ***kwargs*)

Argparse Action that prints a program copyright

option_strings
list – list of str giving command line flags that call this action

dest
str – Namespace reference to value

nargs
bool – True if multiple arguments specified

****kwargs**
various – optional arguments to pass to super call

It does not escape my notice that this action prints arbitrary text without any sort of "copyright-specific" attributes or mangling. This function is only called this for readability in code.

__call__(*parser*, *namespace*, *value*, *option_string=None*)
Prints the given text stripped of excess whitespace and exits

Parameters

- **parser** (*ArgumentParser*) – parser used to generate values
- **namespace** (*Namespace*) – namespace to set values for
- **value** (*str*) – actual value specified by user
- **option_string** (*str*) – argument flag used to call this function

Raises

- `TypeError` – if value is not a string
- `ValueError` – if value cannot, for any reason, be parsed by commas

3.1.4 Open

`Open` is an `argparse action` that seamlessly handles reading and writing compressed files using the gzip, bz2, and lzma libraries. To do this, `Open` actually exposes the arguments of each to libraries `*File` function to the command line after automatically selecting the proper library based on the arguments it receives. Essentially, this `action` operates in a read mode and a write/append mode. In read mode, when mode is equal to any read mode supported by the appropriate library such as `r` or `rb`, `Open` reads the first few bytes of the file to see what compression format the file uses and then opens the file with the corresponding in decompression algorithm. In write mode, basically when mode is set to anything else, `Open` just checks the file extension and maps it to the corresponding compression algorithm. If `Open` does not recognize the first few bytes of a file or a file extension, it defaults to reading and writing in plain text.

As aforementioned, `Open` exposes the arguments of the underlying library. It does this by collecting arbitrary arguments, filtering them by the supported arguments of the `*File` functions, and only passing those arguments to the function. For example, `GzipFile` and `BZ2File` can control the level on compression via the argument `compresslevel` while `LZMAFile` uses `preset` to control compression levels. In order to use these arguments at the `argparse` level, simply add them as options to `Open` as follows:

```
from arandomness.argparse import Open
import argparse
parser = argparse.ArgumentParser(description=__doc__)
parser.add_argument('--gzip',
                    action=Open,
                    mode='r',
                    type=str,
                    compresslevel=9,
                    help='compressed file to read')
```

```
parser.add_argument('--bz2',
                    action=Open,
                    mode='w',
                    type=str,
                    compresslevel=9
                    help='compressed file to write')
parser.add_argument('--lzma',
                    action=Open,
                    mode='w',
                    type=str,
                    preset=9,
                    help='compressed file to write')
args = parser.parse_args(['-i', 'input.gz', '-o', 'output.xz'])
```

As stated, this works for any argument and arguments that aren't supported by the `*File` are silently ignored.

Common use example:

```
from arandomness.argparse import Open
import argparse
parser = argparse.ArgumentParser(description=__doc__)
parser.add_argument('-i', '--input',
                    action=Open,
                    mode='r',
                    type=str,
                    help='compressed file to read')
parser.add_argument('-o', '--output',
                    action=Open,
                    mode='w',
                    type=str,
                    help='compressed file to write')
args = parser.parse_args(['-i', 'input.gz', '-o', 'output.xz'])
```

API Documentation

class `arandomness.argparse.`**Open**(*option_strings*, *dest*, *mode='rb'*, *nargs=None*, ***kwargs*)

Argparse Action that detects and opens compressed files for rw

option_strings
list – list of str giving command line flags that call this action

dest
str – Namespace reference to value

mode
str – mode to pass to (de)compression algorithm

nargs
bool – True if multiple arguments specified

****kwargs**
various – optional arguments to pass to argparse and algo

__call__(*parser*, *namespace*, *value*, *option_string=None*, ***kwargs*)

Detects and opens compressed files

Parameters

- **parser** (*ArgumentParser*) – parser used to generate values

- **namespace** (*Namespace*) – namespace to set values for
- **value** (*str*) – actual value specified by user
- **option_string** (*str*) – argument flag used to call this function
- ****kwargs** (*various*) – optional arguments later passed to the compression algorithm

3.1.5 ParseCommas

By default, `argparse` parses multiple arguments by spaces. While useful, it can sometimes be more practical, or at least easier to read, arguments parsed by commas when multiple arguments make use of nargs. `ParseCommas` simply takes a string, splits it by commas, and sets the resulting list as the value for the argument. For example:

```
from arandomness.argparse import ParseCommas
import argparse
parser = argparse.ArgumentParser(description=__doc__)
parser.add_argument('-a', '--an_argument',
                    action=ParseCommas,
                    type=str,
                    help='nargs using a string')
args = parser.parse_args(['hello,world'])
print(args.an_argument)
```

So the argument `hello,world` would be set as `['hello', 'world']` in args.

API Documentation

class `arandomness.argparse.`**ParseCommas**(*option_strings*, *dest*, *nargs=None*, ***kwargs*)

Argparse Action that parses arguments by commas

option_strings
list – list of str giving command line flags that call this action

dest
str – Namespace reference to value

nargs
str – number of args as special char or int

****kwargs**
various – optional arguments to pass to super call

__call__(*parser*, *namespace*, *value*, *option_string=None*)

Called by Argparse when user specifies a comma-separated list

Simply splits a list by commas and adds the values to namespace.

Parameters

- **parser** (*ArgumentParser*) – parser used to generate values
- **namespace** (*Namespace*) – namespace to set values for
- **value** (*str*) – actual value specified by user
- **option_string** (*str*) – argument flag used to call this function

Raises

- `TypeError` – if value is not a string

- `ValueError` – if value cannot, for any reason, be parsed by commas

3.2 string

Initializes string package of arandomness

3.2.1 Introduction

The `string` subpackage of arandomness contains a couple functions that analyze or manipulate strings in some way. That's about as specific as this subpackage gets. Enjoy!

3.2.2 autocorrect

The `autocorrect` function takes a single query string and a list of "correct" strings and identifies which string in the list the query most closely matches. There are many far more robust autocorrect algorithms written in Python than this one, but they all require a list of words organized by their frequency in a given language. Basically, these autocorrect algorithms are aimed at correcting words specific to a language and are thus better suited for use in language processing software, e.g. texting apps. This algorithm uses any list of strings and is order-agnostic. Thus, my `autocorrect` is better suited for attempting to match queries to small lists of arbitrary strings.

To help realize this concept, I have used this function in a program that presented data in a database about programs available on a given system. The query was the user's request and the possible strings was simply the list of program names in the database. Thus, if a user misspelled a program name, the program likely produced the proper entry.

API Documentation

`arandomness.string.`**`autocorrect`**(*query*, *possibilities*, *delta=0.75*)

Attempts to figure out what possibility the query is

This autocorrect function is rather simple right now with plans for later improvement. Right now, it just attempts to finish spelling a word as much as possible, and then determines which possibility is closest to said word.

Parameters

- **`query`** (*`unicode`*) – query to attempt to complete
- **`possibilities`** (*`list`*) – list of unicodes of possible answers for query
- **`delta`** (*`float`*) – minimum delta similarity between query and any given possibility for possibility to be considered. Delta used by difflib.get_close_matches().

Returns best guess of correct answer

Return type unicode

Raises `AssertionError` – raised if no matches found

Example

```
>>> autocorrect('bowtei', ['bowtie2', 'bot'])
'bowtie2'
```

3.2.3 max_substring

The `max_substring` function takes in a list of strings and finds the longest substring that they all share. By default, `max_substring` starts at the beginning of each string, but it can be optionally start at a later position as demonstrated in the docstring examples.

API Documentation

`arandomness.string.`**`max_substring`**(*words*, *position=0*, *_last_letter=''*)

Finds max substring shared by all strings starting at position

Parameters

- **`words`** (*`list`*) – list of unicode of all words to compare
- **`position`** (*`int`*) – starting position in each word to begin analyzing for substring
- **`_last_letter`** (*`unicode`*) – last common letter, only for use internally unless you really know what you are doing

Returns max string common to all words

Return type unicode

Examples

```
>>> max_substring(['aaaa', 'aaab', 'aaac'])
'aaa'
>>> max_substring(['abbb', 'bbbb', 'cbbb'], position=1)
'bbb'
>>> max_substring(['abc', 'bcd', 'cde'])
''
```

3.3 trees

Initializes trees package of arandomness

Copyright: __init__.py Initializes trees package of arandomness Copyright (C) 2017 Alex Hyer

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

3.3.1 Introduction

The `argparse` subpackage of arandomness contains scripts and classes relating to trees.

3.3.2 OmniTree (Deprecated)

OmniTree is a class for creating a tree where each node can contain multiple children and multiple parents. I began writing this class because I could not find an extant tree library that supported this many-to-many paradigm. While binary and/or hierarchical tree with only a single parent are common, see anytree, trees like OmniTree are not. After a lot of R&D, I realized that these is such a structure, a graph. Basically, many-to-many trees don't exist because they cannot by definition. Thus, OmniTree is pointless as there are already amazing libraries for manipulating and managing graphs, such as NetworkX. As such, OmniTree is deprecated and is only included for archival purposes.

API Documentation

class `arandomness.trees.`**`OmniTree`**(*label=None*, *children=None*, *parents=None*)
A many-to-many tree for organizing and manipulating hierarchical data

`label`
unicode – optional, arbitrary name for node

`__init__`(*label=None*, *children=None*, *parents=None*)
Initialize node and inform connected nodes

`__weakref__`
list of weak references to the object (if defined)

`add_children`(*children*)
Adds new children nodes after filtering for duplicates

Parameters **`children`** (*list*) – list of OmniTree nodes to add as children

`add_parents`(*parents*)
Adds new parent nodes after filtering for duplicates

Parameters **`parents`** (*list*) – list of OmniTree nodes to add as parents

`find_branches`(*labels=False*, *unique=False*)
Recursively constructs a list of pointers of the tree's structure

Parameters

- **`labels`** (*bool*) – If True, returned lists consist of node labels. If False (default), lists consist of node pointers. This option is mostly intended for debugging purposes.
- **`unique`** (*bool*) – If True, return lists of all unique, linear branches of the tree. More accurately, it returns a list of lists where each list contains a single, unique, linear path from the calling node to the tree's leaf nodes. If False (default), a highly-nested list is returned where each nested list represents a branch point in the tree. See Examples for more.

Examples

```
>>> from arandomness.trees import OmniTree
>>> a = OmniTree(label='a')
>>> b = OmniTree(label='b', parents=[a])
>>> c = OmniTree(label='c', parents=[b])
>>> d = OmniTree(label='d', parents=[b])
>>> e = OmniTree(label='e', parents=[c, d])
>>> a.find_branches(labels=True)
['a', ['b', ['c', ['e']], ['d', ['e']]]]
>>> a.find_branches(labels=True, unique=True)
[['a', 'b', 'c', 'e'], ['a', 'b', 'd', 'e']]
```

find_loops(*_path=None*)

Crappy function that finds a single loop in the tree

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

CHAPTER 5

Copyright

arandomness operates under the `GPLv3 License` and may be edited and redistributed as per that license.

Python Module Index

a

arandomness, 1
arandomness.argparse, 7
arandomness.string, 12
arandomness.trees, 13

Index

Symbols

__call__() (arandomness.argparse.CheckThreads method), 8
__call__() (arandomness.argparse.CopyRight method), 9
__call__() (arandomness.argparse.Open method), 10
__call__() (arandomness.argparse.ParseCommas method), 11
__init__() (arandomness.trees.OmniTree method), 14
__weakref__ (arandomness.trees.OmniTree attribute), 14

A

add_children() (arandomness.trees.OmniTree method), 14
add_parents() (arandomness.trees.OmniTree method), 14
arandomness (module), 1
arandomness.argparse (module), 7
arandomness.string (module), 12
arandomness.trees (module), 13
autocorrect() (in module arandomness.string), 12

C

CheckThreads (class in arandomness.argparse), 8
CopyRight (class in arandomness.argparse), 8

D

dest (arandomness.argparse.CopyRight attribute), 9
dest (arandomness.argparse.Open attribute), 10
dest (arandomness.argparse.ParseCommas attribute), 11

F

find_branches() (arandomness.trees.OmniTree method), 14
find_loops() (arandomness.trees.OmniTree method), 15

L

label (arandomness.trees.OmniTree attribute), 14

M

max_substring() (in module arandomness.string), 13
mode (arandomness.argparse.Open attribute), 10

N

nargs (arandomness.argparse.CopyRight attribute), 9
nargs (arandomness.argparse.Open attribute), 10
nargs (arandomness.argparse.ParseCommas attribute), 11

O

OmniTree (class in arandomness.trees), 14
Open (class in arandomness.argparse), 10
option_strings (arandomness.argparse.CopyRight attribute), 8
option_strings (arandomness.argparse.Open attribute), 10
option_strings (arandomness.argparse.ParseCommas attribute), 11

P

ParseCommas (class in arandomness.argparse), 11